FIG. 1

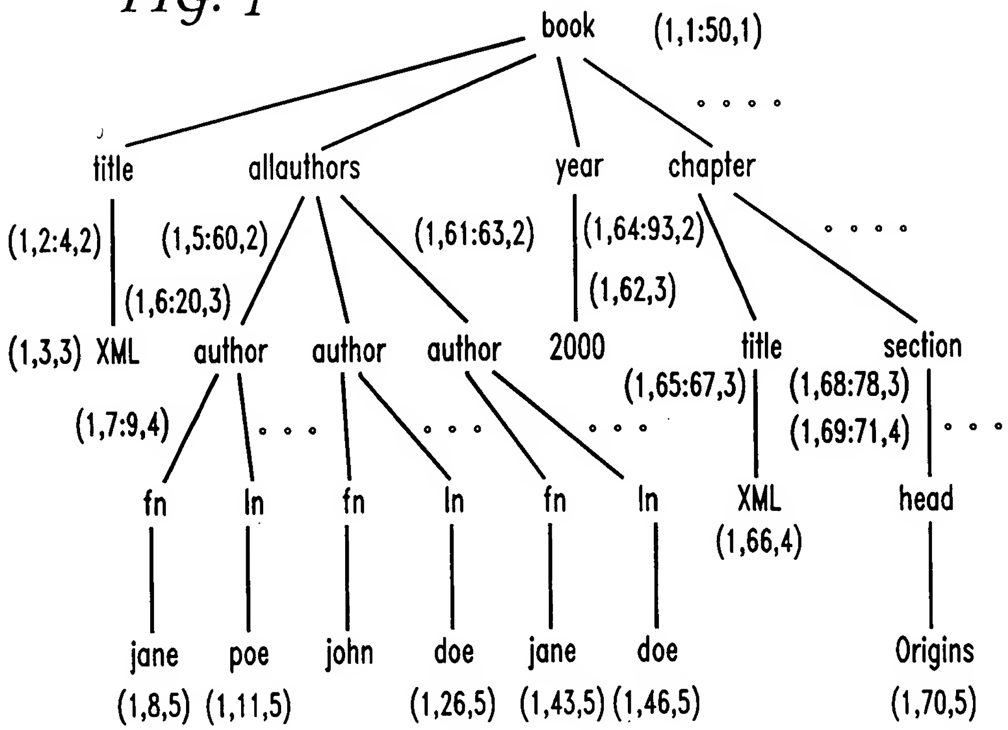


FIG. 2A

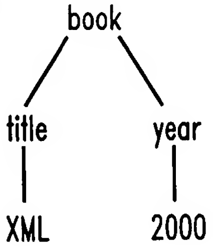


FIG. 2B

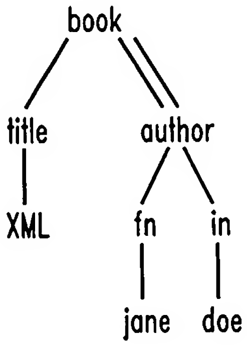
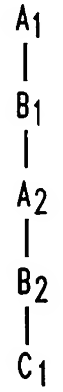
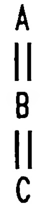


FIG. 3A



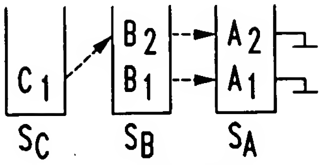
DATA

FIG. 3B



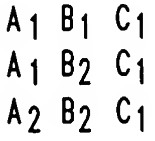
QUERY

FIG. 3C



STACK ENCODING

FIG. 3D



QUERY RESULTS

FIG. 4

```
Algorithm PathStack(q)
  while ¬end(q)
    qmin = getMinSource(q)
    for q_i in subtreeNodes(q)  // clean stacks
      while (¬empty(S_{q_i}) ∧ topB(S_{q_i}) < nextL(T_{q_min}))
        pop(S_{q_i})
    moveStreamToStack(T_{q_min}, S_{q_min}, pointer to
                                  top(S_{parent(q_min)}))
    if (isleaf(q_min))
      showSolutions(S_{q_min},1)
      pop (S_{q_min})
Function end(q)
  return ∀q_i ∈ subtreeNodes(q): isLeaf(q_i) ⇒ eof(T_{q_i})

Function getMinSource(q)
  return q_i ∈ subtreeNodes(q) such that nextL(T_{q_i})
    is minimal
Procedure moveStreamToStack(T_q,S_q,p)
  push(S_q,(next(T_q),p))
  advance(T_q)
```

PathStack

FIG. 4A

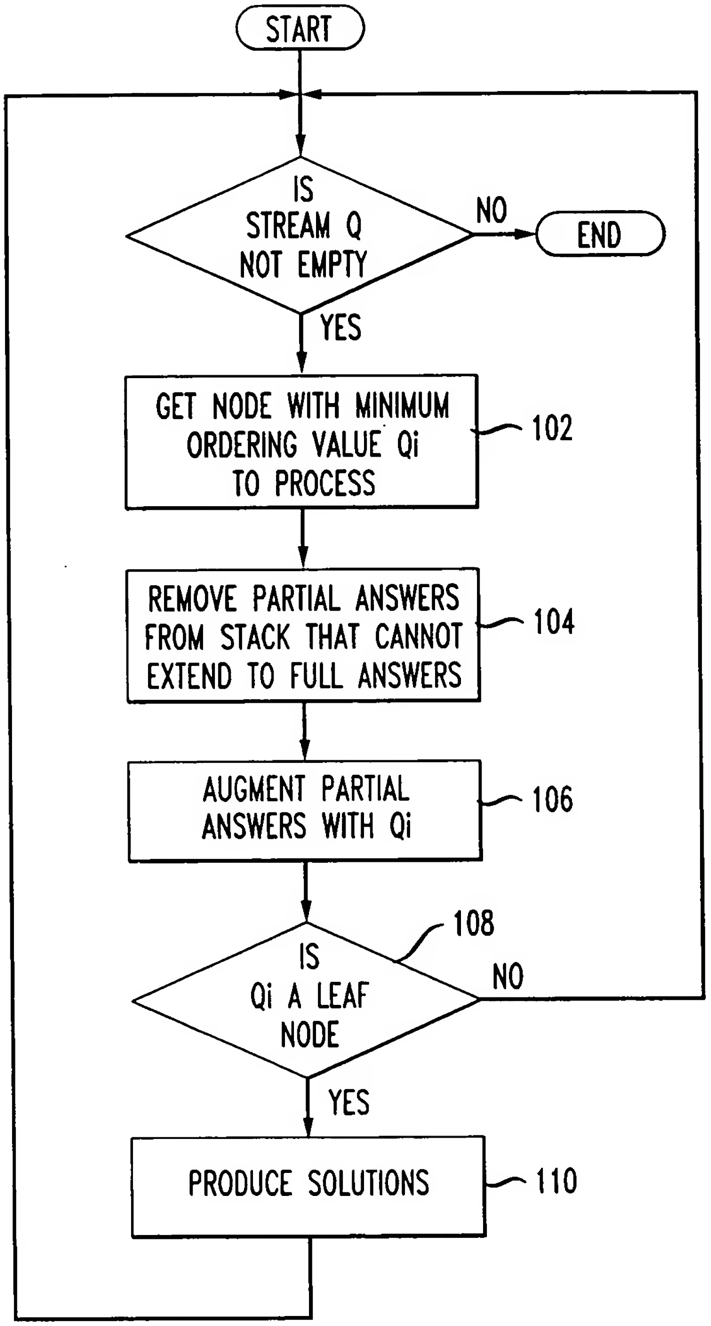

START
IS STREAM Q NOT EMPTY
NO
END
YES
GET NODE WITH MINIMUM ORDERING VALUE Qi TO PROCESS
102
REMOVE PARTIAL ANSWERS FROM STACK THAT CANNOT EXTEND TO FULL ANSWERS
104
AUGMENT PARTIAL ANSWERS WITH Qi
106
108
IS Qi A LEAF NODE
NO
YES
PRODUCE SOLUTIONS
110

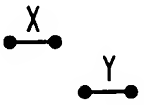
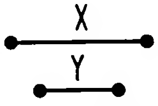
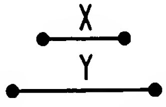
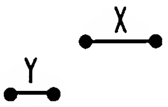
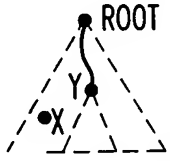
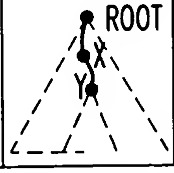
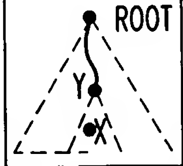
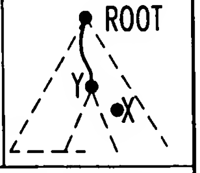
FIG. 5

```
Procedure showSolutions(SN,SP)
// Assume, for simplicity, that the stack of the query
// nodes from the root to the current leaf node we
// are interested in can be accessed as S[1],...,S[n].
// Also assume that we have a global array index[1..n]
//    of pointers to the stack elements.
// index[i] represents the position in the i'th stack that
// we are interested in for the current solution, where
// the bottom of each stack has position 1

// Mark we are interested in position SP of stack SN.
index[SN] = SP
if (SN == 1) // we are in the root
    // output solutions from the stacks
    output (S[n].index[n],...,S[1].index[1])
  else // recursive call
      for i = 1 to S[SN].index[SN].pointer_to_parent
          showSolutions(SN - 1,i)
```

Procedure showSolutions

FIG. 6

	CASE 1	CASE 1	CASE 1	CASE 1
PROPERTY	X.R<Y.L	X.L<Y.L X.R>Y.R	X.L>Y.L X.R<Y.R	X.L>Y.R
SEGMENTS				
TREE				

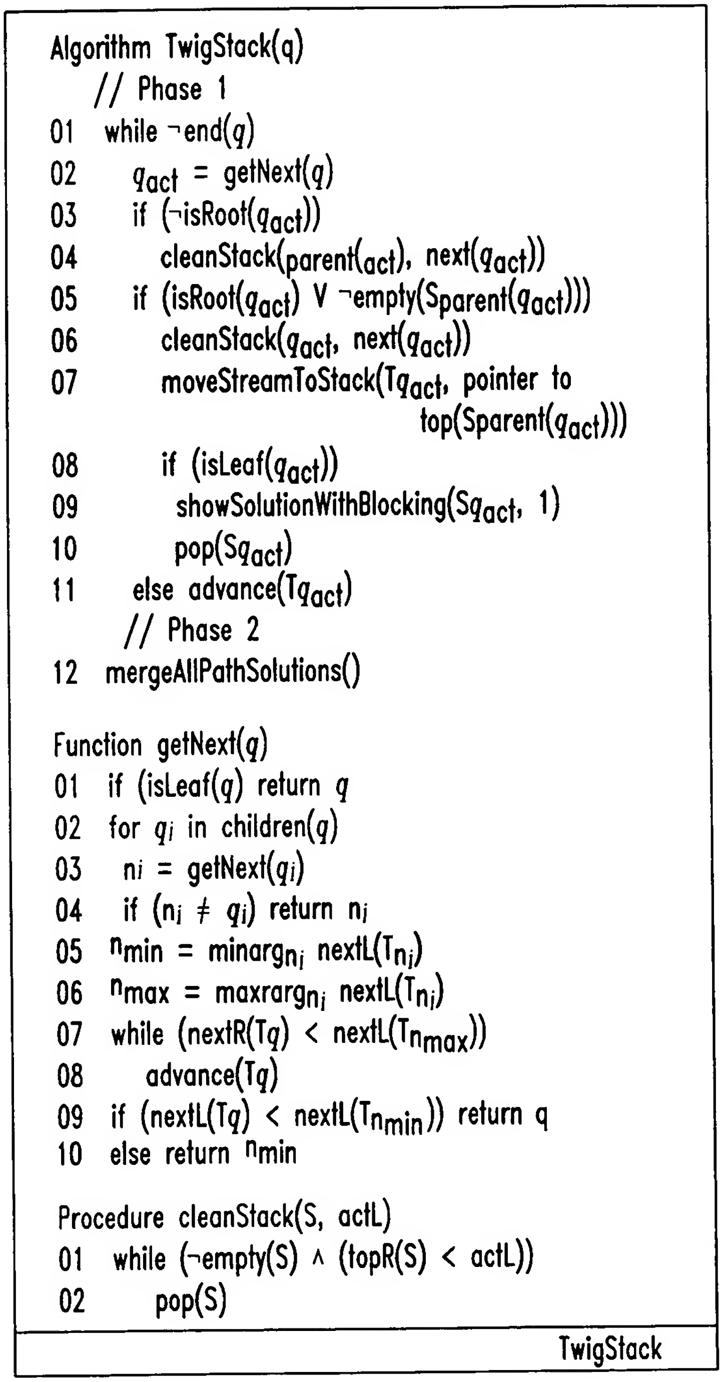
CASES FOR PathStack AND TwigStack

FIG. 7

```
Algorithm PathMPMJ($q$)
while ($\neg$ eof($T_q$) $\wedge$ (isRoot($q$) $\vee$
                      nextL($q$) < nextR(parent($q$))))
  for ($q_i \in$ subtreeNodes($q$)) // advance descendants
    while (nextL($q_i$) < nextL(parent)$q_i$)))
      advance($T_{q_i}$)
    PushMark($T_{q_i}$)
  if (isLeaf($q$)) // solution in the streams' heads
    outputSolution()
  else PathMPMJ(child($q$))
  advance($T_q$)
  for ($q_i \in$ subtreeNodes($q$)) // backtrack descendants
    PopMark($T_{q_i}$)
```

PathMPMJ

FIG. 8



```
Algorithm TwigStack(q)
  // Phase 1
  while ¬end($q$)
    $q_{act}$ = getNext($q$)
    if (¬isRoot($q_{act}$))
      cleanStack(parent($_{act}$), next($q_{act}$))
    if (isRoot($q_{act}$) V ¬empty(S$_{parent(q_{act})}$))
      cleanStack($q_{act}$, next($q_{act}$))
      moveStreamToStack(T$_{q_{act}}$, pointer to
                                top(S$_{parent(q_{act})}$))
      if (isLeaf($q_{act}$))
        showSolutionWithBlocking(S$_{q_{act}}$, 1)
        pop(S$_{q_{act}}$)
    else advance(T$_{q_{act}}$)
    // Phase 2
  mergeAllPathSolutions()

Function getNext($q$)
  if (isLeaf($q$) return $q$
  for $q_i$ in children($q$)
    $n_i$ = getNext($q_i$)
    if ($n_i \neq q_i$) return $n_i$
  $n_{min}$ = minarg$_{n_i}$ nextL(T$_{n_i}$)
  $n_{max}$ = maxrarg$_{n_i}$ nextL(T$_{n_i}$)
  while (nextR(T$_q$) < nextL(T$_{n_{max}}$))
    advance(T$_q$)
  if (nextL(T$_q$) < nextL(T$_{n_{min}}$)) return q
  else return $n_{min}$

Procedure cleanStack(S, actL)
  while (¬empty(S) ∧ (topR(S) < actL))
    pop(S)
```

TwigStack

FIG. 8A

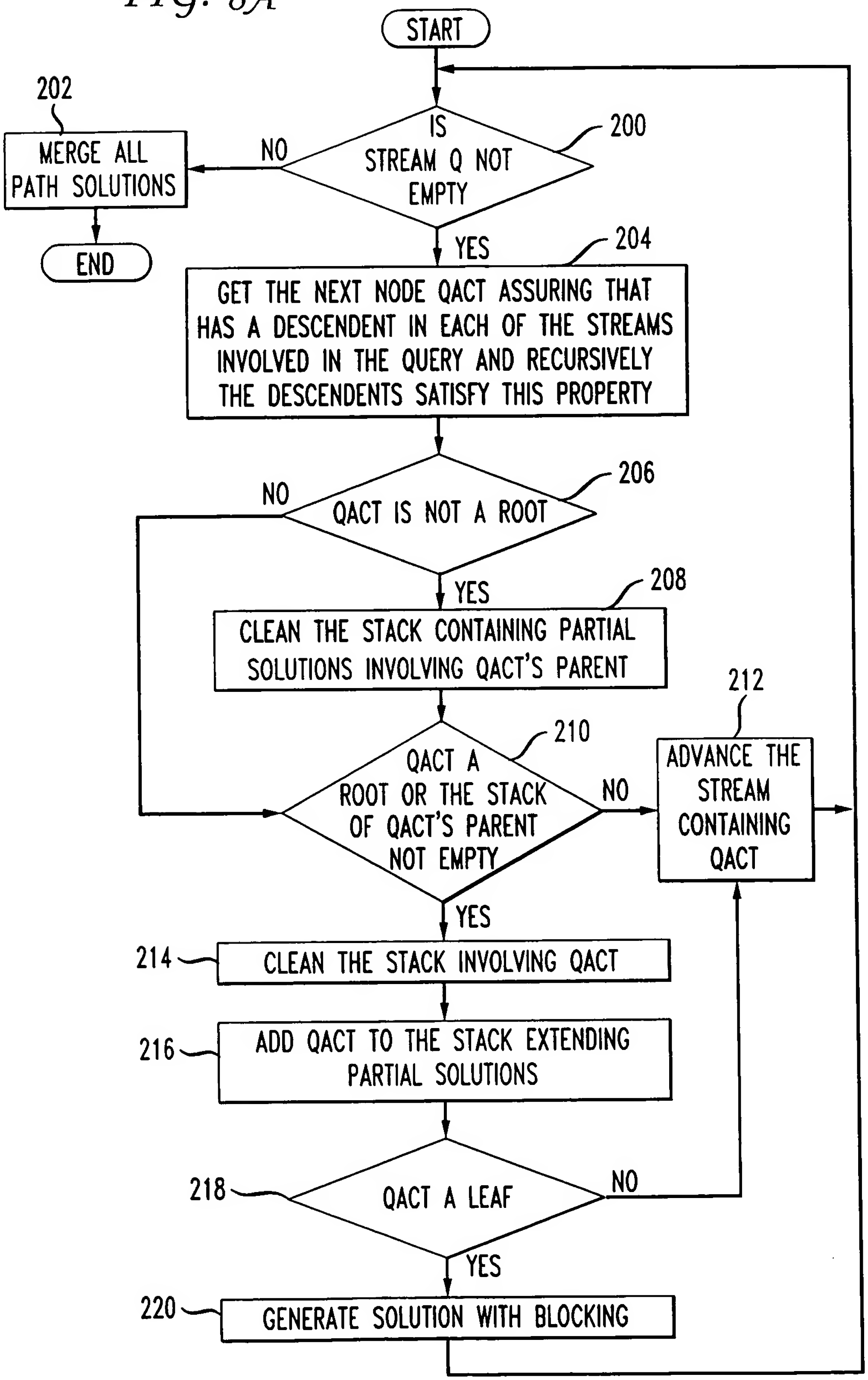

START
IS STREAM Q NOT EMPTY
200
NO
YES
202
MERGE ALL PATH SOLUTIONS
END
204
GET THE NEXT NODE QACT ASSURING THAT HAS A DESCENDENT IN EACH OF THE STREAMS INVOLVED IN THE QUERY AND RECURSIVELY THE DESCENDENTS SATISFY THIS PROPERTY
206
QACT IS NOT A ROOT
NO
YES
208
CLEAN THE STACK CONTAINING PARTIAL SOLUTIONS INVOLVING QACT'S PARENT
210
QACT A ROOT OR THE STACK OF QACT'S PARENT NOT EMPTY
NO
YES
212
ADVANCE THE STREAM CONTAINING QACT
214
CLEAN THE STACK INVOLVING QACT
216
ADD QACT TO THE STACK EXTENDING PARTIAL SOLUTIONS
218
QACT A LEAF
NO
YES
220
GENERATE SOLUTION WITH BLOCKING

FIG. 9

```
Algorithm TwigStackXB(q)
01   while ¬end($q$)
02      $q_{act}$ = getNext($q$)
(03)    if (isPlainValue($Tq_{act}$))
04         if (¬ isRoot($q_{act}$))
05            cleanStack(parent ($q_{act}$), next($q_{act}$))
06         if (isRoot($q_{act}$) V ¬empty($S_{parent}$ ($q_{act}$)))
07            cleanStack($q_{act}$, next($q_{act}$))
08            moveStreamToStack($Tq_{act}$, pointer to
                                 top($S_{parent}$ ($q_{act}$)))
09            if (isLeaf($q_{act}$))
10               showSolutionsWithBlocking($Sq_{act}$,1)
11               pop($Sq_{act}$)
12         else advance($Tq_{act}$)
(13)    else  if (¬isRoot($q_{act}$) Λ empty($S_{parent}$($q_{act}$)) Λ
                 nextL($T_{parent}$($q_{act}$)) > nextR($Tq_{act}$))
(14)    advance($Tq_{act}$) // Not part of a solution
(15)    else // Might have a child in some solution
(16)    drillDown($Tq_{act}$)
      // Phase 2
17   mergeAllPathSolutions()

Function getNext($q$)
01   if (isLeaf($q$) return $q$
02   for $q_i$ in children($q$)
03      $n_i$= getNext($q_i$)
(04)  if ($q_i \neq n_i$ V ¬ isPlainValue($Tn_i$)) return $n_i$
05   $n_{min}$ = minarg$_{n_i}$ nextL($Tn_i$)
06   $n_{max}$ = maxrarg$_{n_i}$ nextL($Tn_i$)
07   while (nextR($Tq$) < nextL($Tn_{max}$))
08      advance($Tq$)
09   if (nextL($Tq$) < nextL($Tn_{min}$)) return q
10   else return $n_{min}$

Procedure cleanStack(S, actL)
01   while (¬ empty(S) ∧ (topR(S) < actL))
02      pop(S)
```

TwigStack

FIG. 9A

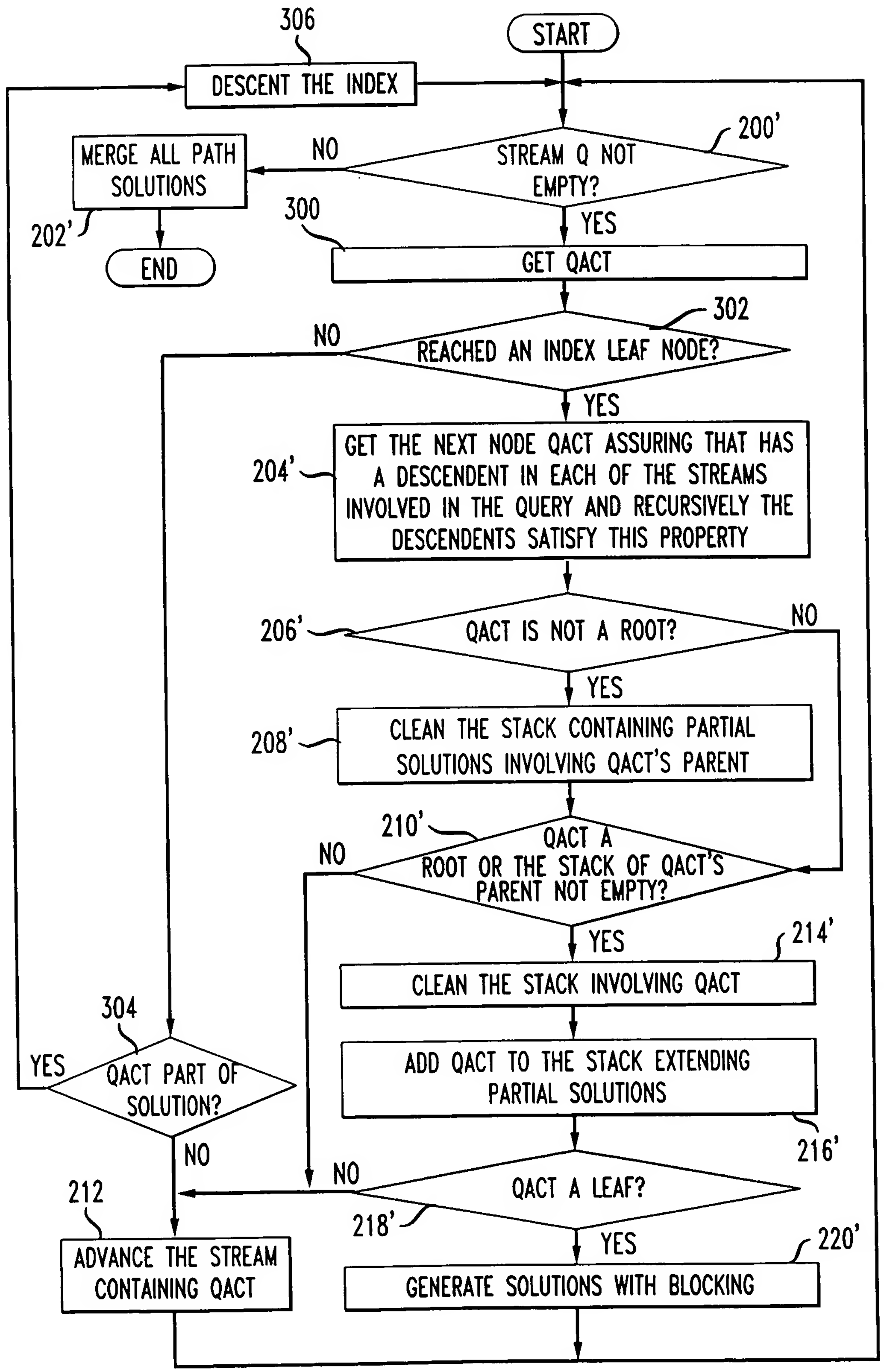

306
START
DESCENT THE INDEX
200'
STREAM Q NOT EMPTY?
NO
MERGE ALL PATH SOLUTIONS
202'
END
300
YES
GET QACT
302
REACHED AN INDEX LEAF NODE?
NO
YES
204'
GET THE NEXT NODE QACT ASSURING THAT HAS A DESCENDENT IN EACH OF THE STREAMS INVOLVED IN THE QUERY AND RECURSIVELY THE DESCENDENTS SATISFY THIS PROPERTY
206'
QACT IS NOT A ROOT?
NO
YES
208'
CLEAN THE STACK CONTAINING PARTIAL SOLUTIONS INVOLVING QACT'S PARENT
210'
QACT A ROOT OR THE STACK OF QACT'S PARENT NOT EMPTY?
NO
YES
214'
CLEAN THE STACK INVOLVING QACT
304
ADD QACT TO THE STACK EXTENDING PARTIAL SOLUTIONS
YES
QACT PART OF SOLUTION?
NO
216'
212
NO
QACT A LEAF?
218'
YES
220'
ADVANCE THE STREAM CONTAINING QACT
GENERATE SOLUTIONS WITH BLOCKING

FIG. 10

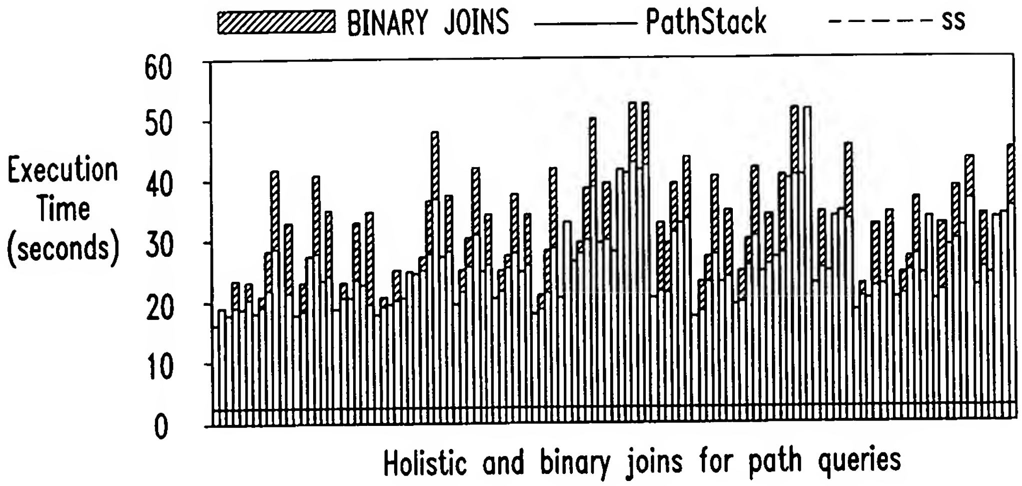

BINARY JOINS
PathStack
ss
60
50
40
30
20
10
0
Execution Time (seconds)
Holistic and binary joins for path queries

FIG. 11

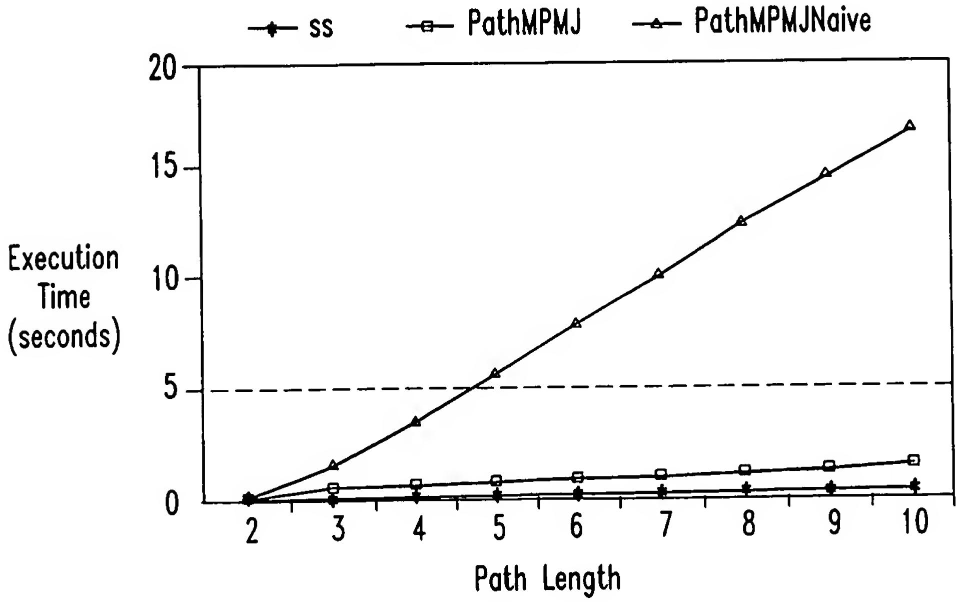
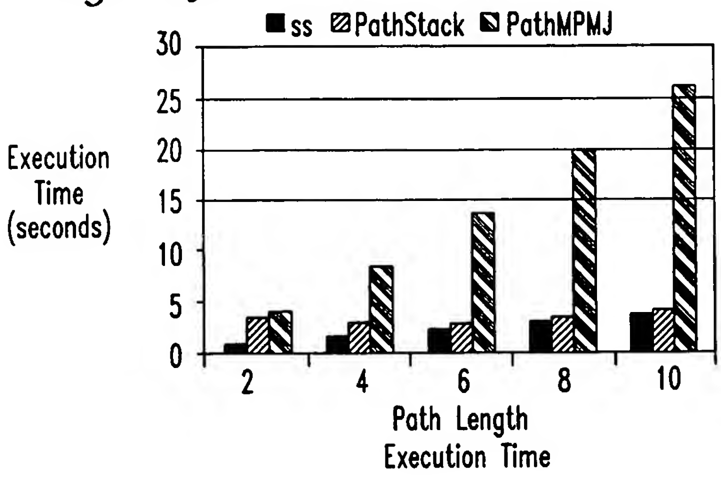
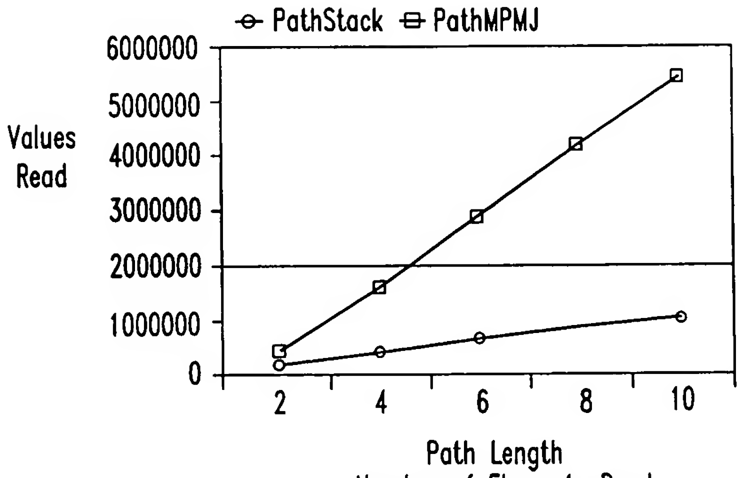

ss
PathMPMJ
PathMPMJNaive
20
15
10
5
0
Execution Time (seconds)
2
3
4
5
6
7
8
9
10
Path Length

FIG. 12A



Execution Time

FIG. 12B



Number of Elements Read

FIG. 13A

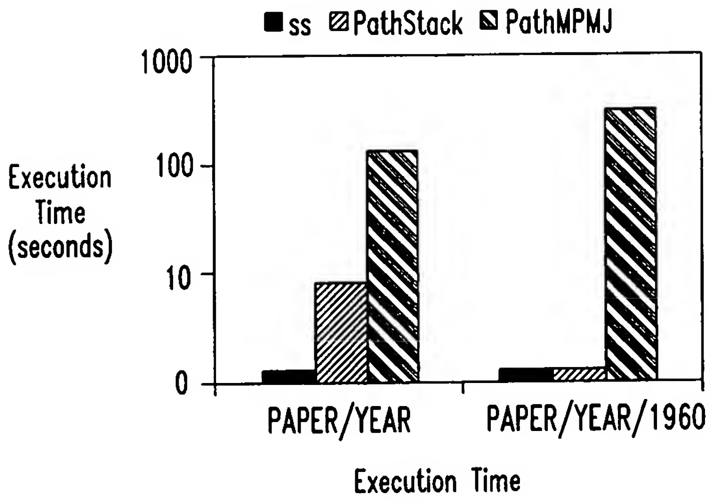


FIG. 13B

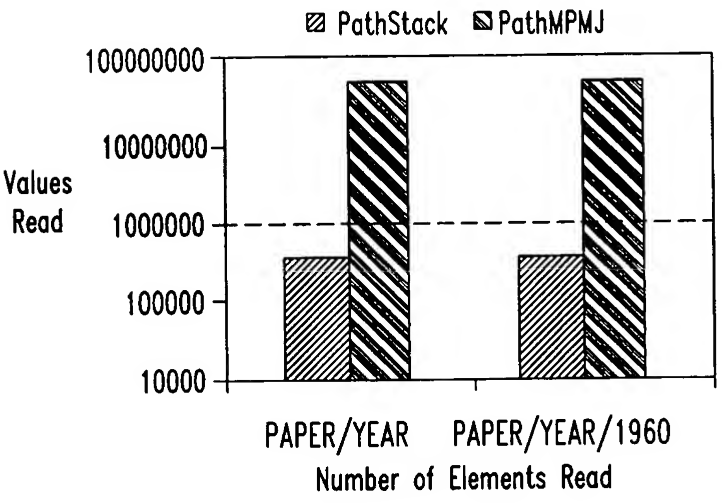


FIG. 14A

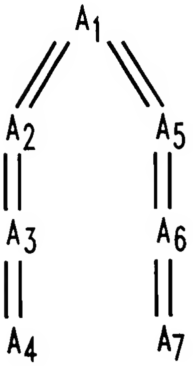


FIG. 14B

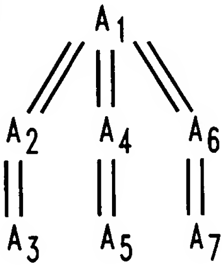


FIG. 14C

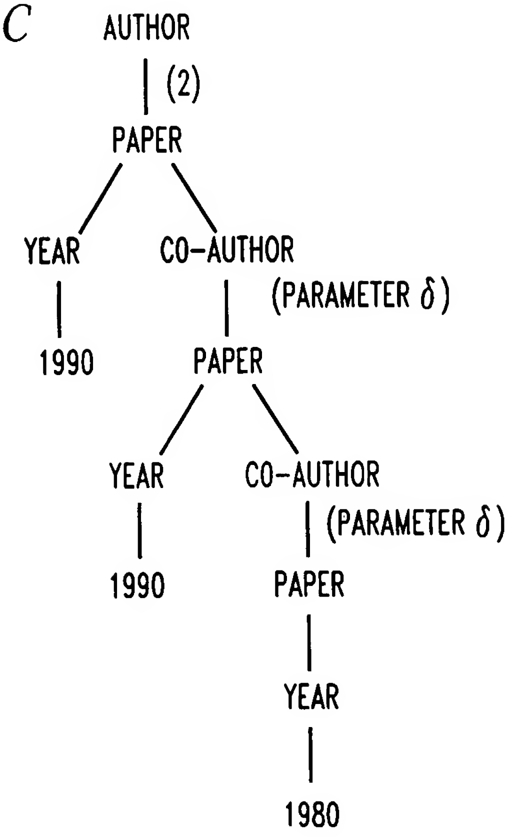
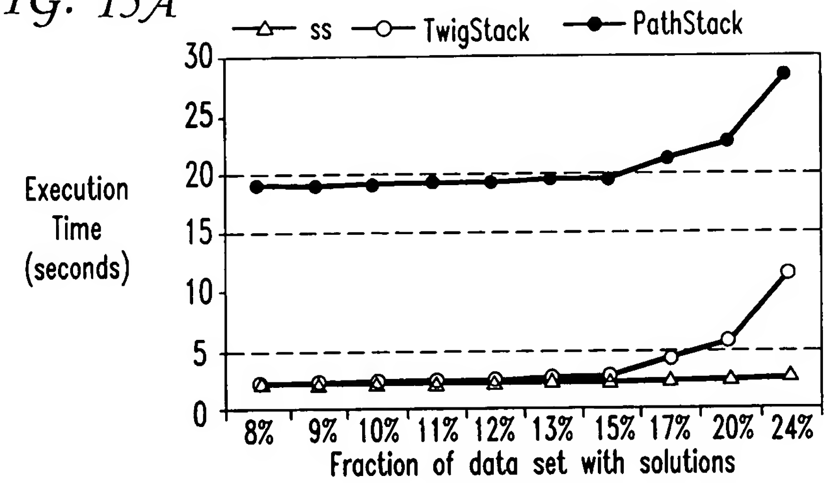
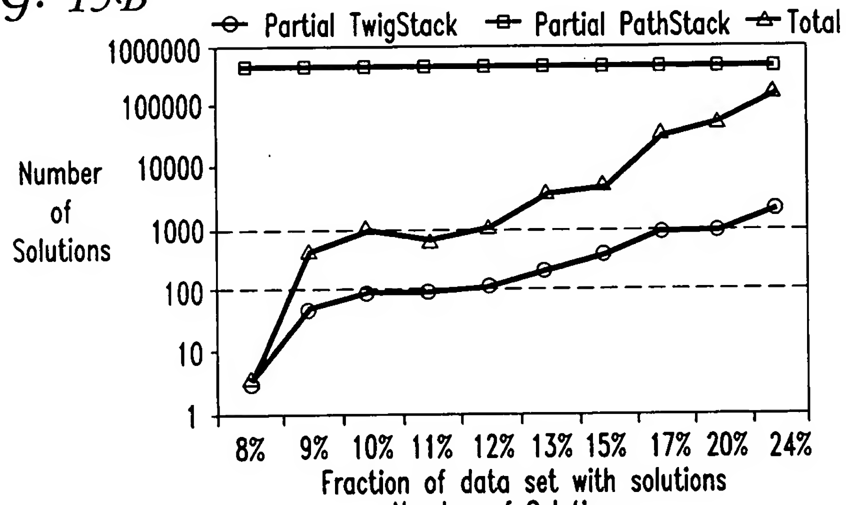


FIG. 15A



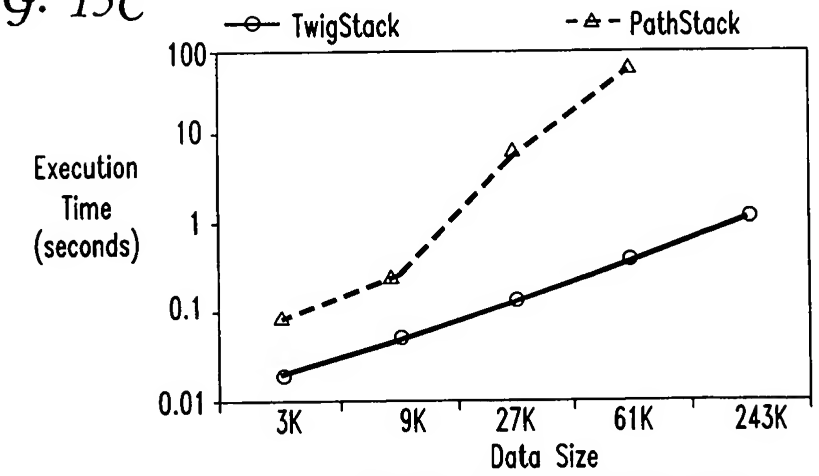
Execution Time

FIG. 15B



Number of Solutions

FIG. 15C



Execution time for complex query

FIG. 16A

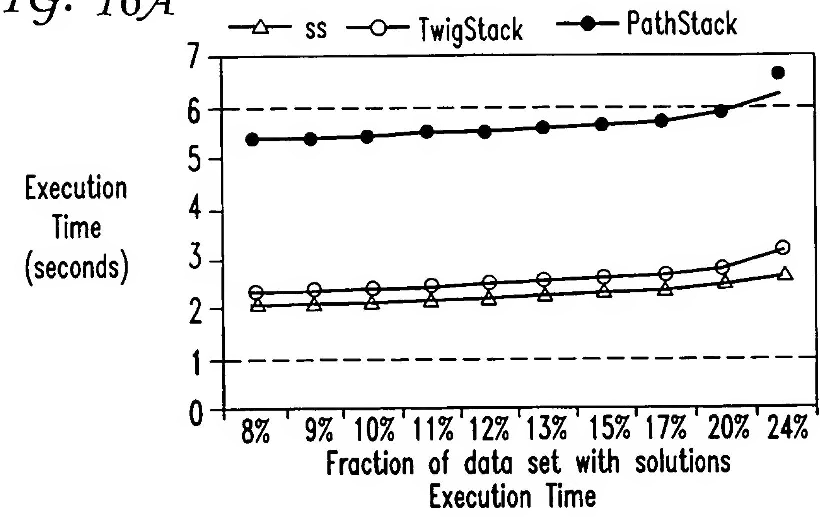


FIG. 16B

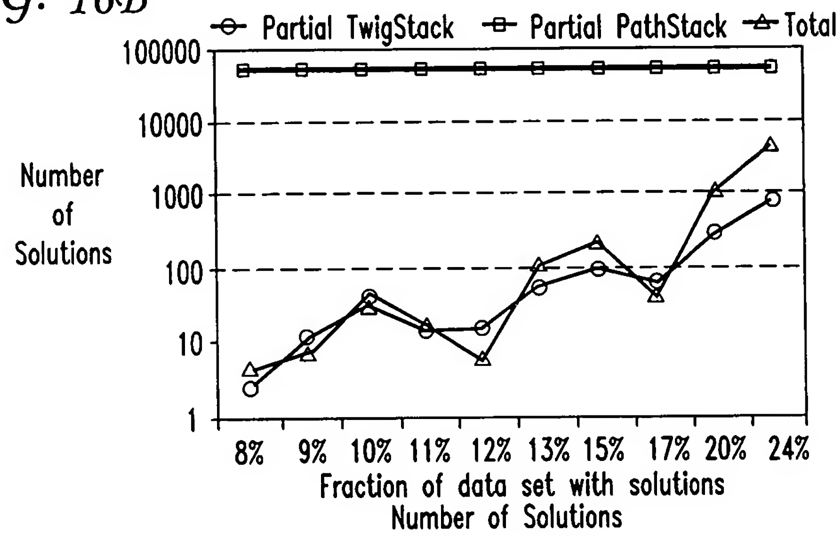


FIG. 16C

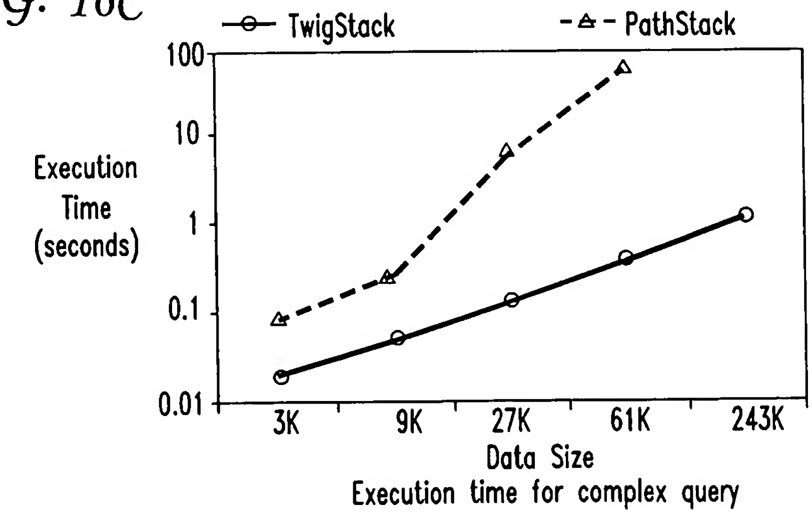
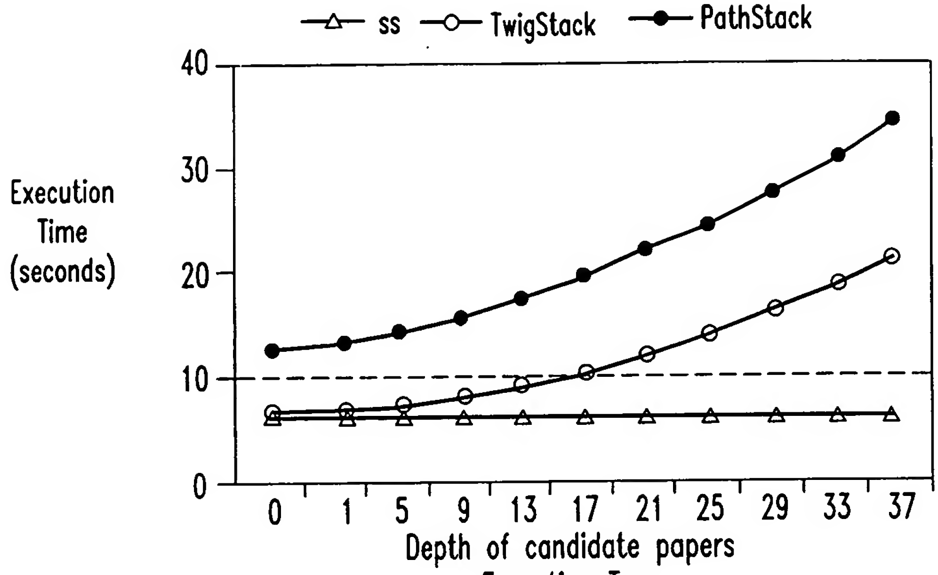
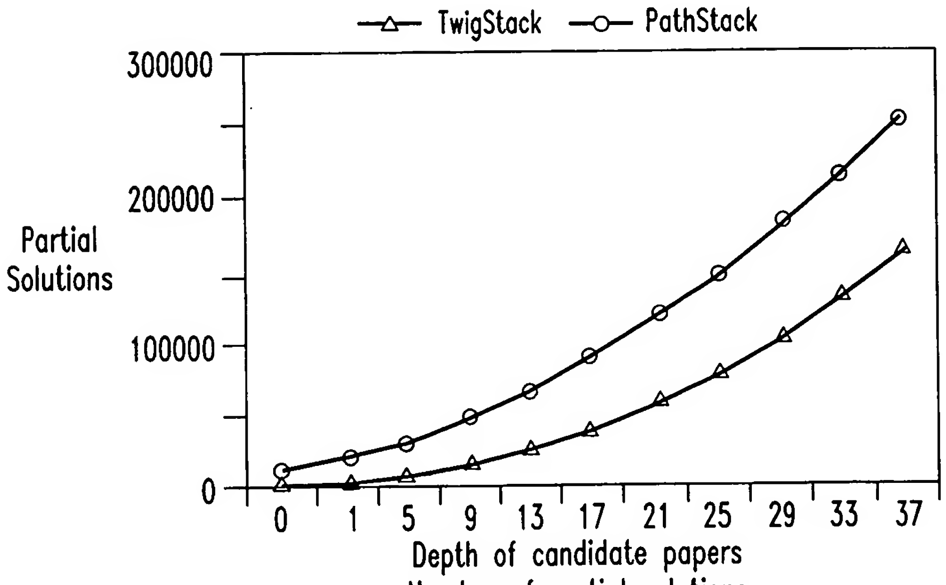


FIG. 17A



Exceution Tme

FIG. 17B



Number of partial solutions

FIG. 18A

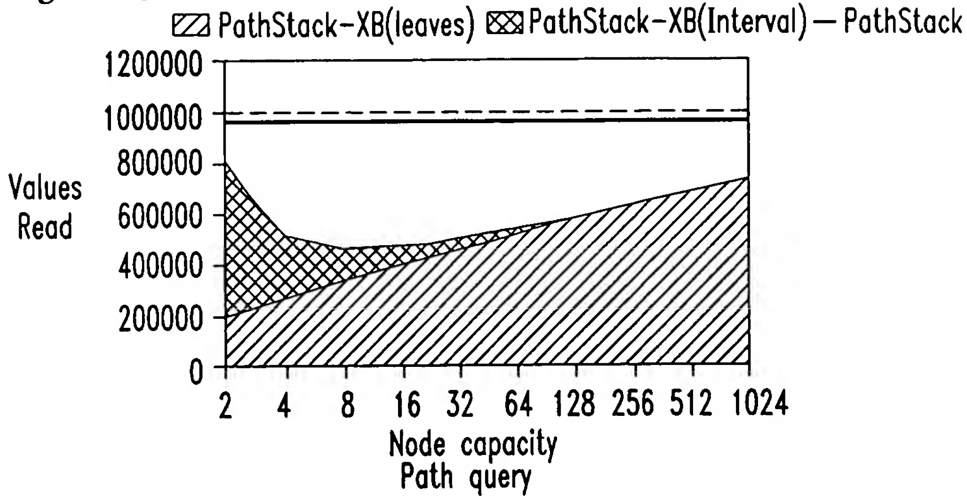


FIG. 18B

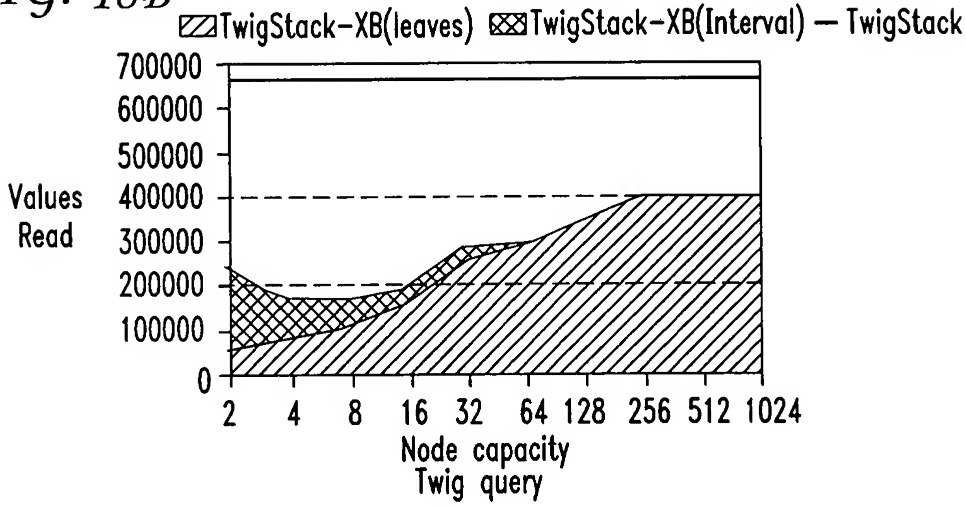
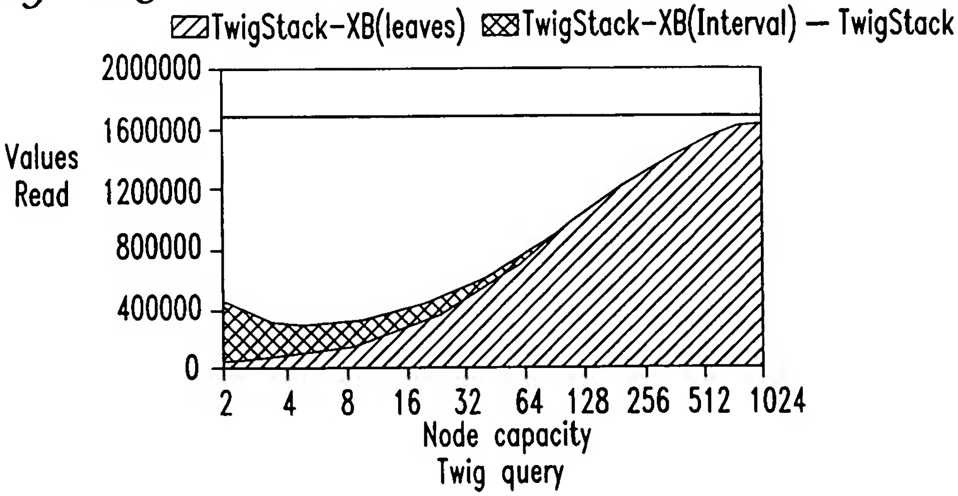


FIG. 18C


▨ TwigStack-XB(leaves)
▩ TwigStack-XB(Interval)
— TwigStack
Values Read
Node capacity
Twig query